AMENDMENTS TO THE CLAIMS

1. (Currently amended) A method of performing high precision emulation of program code instructions for a subject machine on a target machine[[,]] including floating point hardware and integer hardware, the method comprising:

determining ~~if~~ that operands in instructions of the program code for the subject machine require a ~~different~~ higher precision than provided for by the hardware of the target machine; and

applying a floating point emulation algorithm to perform intermediate calculations on the operands of the instructions at a higher precision than the precision supported by the hardware of the target machine to produce intermediate values;

wherein at each stage, the intermediate values are tested to determine whether the intermediate values have reached a point at which the hardware of the target machine has enough precision to finish the calculation without loss of accuracy, such that when it is determined based upon the intermediate calculations that the floating point hardware of the target machine provides sufficient precision to finish the calculations required by the instructions without loss of accuracy, the floating point hardware on the target machine is utilized to finish the calculations; and the integer hardware on the target machine is utilized to perform calculations not selected to be performed by the floating point hardware.

2. (Canceled)

3. (Currently Amended) The method of claim ~~2~~1, wherein the program code instructions are accumulated instructions that are calculated at a higher precision than the operands capable of being handled by the target machine.

4. (Original) The method of claim 3, wherein the program code instructions are floating point accumulated instructions of the form: $d = \pm(a*b \pm c)$,

wherein a, b, c and d are operands expressible as floating point numbers.

5. (Currently Amended) The method of claim 4, further comprising identifying whether any of the operands (a, b, or c) are special values having a known result that all compatible hardware will produce regardless of the level of precision of said <u>compatible</u> hardware.

6. (Original) The method of claim 5, wherein said special values include either zero, infinity, or NaN (not a number), wherein the floating point hardware is utilized to calculate the result of the accumulated instructions when any of the operands (a, b, or c) are identified as special values.

7. (Original) The method of claim 4, wherein said floating point emulation algorithm further comprises:

determining whether the exponent for the result of the multiplication of (a*b) overlaps with the exponent of c; and

utilizing the floating point hardware to calculate the result of the accumulated instructions when the exponent for the result of the multiplication of (a*b) fails to overlap with the exponent of c.

8. (Original) The method of claim 7, wherein, when the exponent for the result of the multiplication of (a*b) overlaps with the exponent of c, said floating point emulation algorithm further comprising:

determining whether the mantissa for the result of the multiplication (a*b) requires more mantissa bits than provided for by said floating point hardware; and

utilizing the floating point hardware to calculate the result of the accumulated instructions when the result of the multiplication (a*b) does not require more mantissa bits than provided for by the floating point hardware.

9. (Original) The method of claim 8, said floating point emulation algorithm further comprising computing the full calculation of a*b using the integer hardware when mantissa for the result of the multiplication (a*b) requires more mantissa bits than provided for by the floating point hardware.

10. (Currently Amended) The method of claim 9, said floating point emulation algorithm further comprising:

determining whether the final resulting mantissa of the mantissa(a*b) - the mantissa (c) equals zero;

utilizing the floating point hardware to make the result of the calculation of a*b + c equal to zero when the resulting mantissa is equal to zero; and

calculating the remaining parts of the calculation of a*b + c using the integer hardware when the final resulting mantissa is not equal to zero.

11. (Currently amended) A computer-readable storage medium having software resident thereon in the form of computer-readable code executable by a computer to perform the following steps in the high precision emulation of program code instructions for a subject machine on a target machine including floating point hardware and integer hardware:

determining ~~if~~ that operands in instructions of the program code for the subject machine require a ~~different~~ higher precision than provided for by the hardware of the target machine; and

applying a floating point emulation algorithm to perform intermediate calculations on the operands of the instructions at a higher precision than the precision supported by the hardware of the target machine to produce intermediate values;

wherein at each stage, the intermediate values are tested to determine whether the intermediate values have reached a point at which the hardware of the target machine has enough precision to finish the calculation without loss of accuracy, such that when it is determined based upon the intermediate calculations that the floating point hardware of the target machine provides sufficient precision to finish the calculations required by the instructions without loss of accuracy, the floating point hardware on the target machine is utilized to finish the calculations; and the integer hardware on the target machine is utilized to perform calculations not selected to be performed by the floating point hardware.

12. (Canceled)

13. (Currently Amended) The computer-readable storage medium of claim ~~12~~11, wherein the program code instructions are accumulated instructions that are calculated at a higher precision than the operands capable of being handled by the target machine.

14. (Original) The computer-readable storage medium of claim 13, wherein the program code instructions are floating point accumulated instructions of the form: $d = \pm (a*b \pm c)$, wherein a, b, c and d are operands expressible as floating point numbers.

15. (Currently Amended) The computer-readable storage medium of claim 14, said computer-readable code further executable for identifying whether any of the operands (a, b, or c) are special values having a known result that all compatible hardware will produce regardless of the level of precision of said compatible hardware.

16. (Original) The computer-readable storage medium of claim 15, wherein said special values include either zero, infinity, or NaN (not a number), wherein the floating point hardware is utilized to calculate the result of the accumulated instructions when any of the operands (a, b, or c) are identified as special values.

17. (Original) The computer-readable storage medium of claim 14, wherein said floating point emulation algorithm further comprises:

determining whether the exponent for the result of the multiplication of (a*b) overlaps with the exponent of c; and

utilizing the floating point hardware to calculate the result of the accumulated instructions when the exponent for the result of the multiplication of (a*b) fails to overlap with the exponent of c.

18. (Original) The computer-readable storage medium of claim 17, wherein, when the exponent for the result of the multiplication of (a*b) overlaps with the exponent of c, said floating point emulation algorithm further comprises:

determining whether the mantissa for the result of the multiplication (a*b) requires more mantissa bits than provided for by said floating point hardware; and

utilizing the floating point hardware to calculate the result of the accumulated instructions when the result of the multiplication (a*b) does not require more mantissa bits than provided for by the floating point hardware.

19. (Original) The computer-readable storage medium of claim 18, said floating point emulation algorithm further comprising computing the full calculation of a*b using the integer hardware when mantissa for the result of the multiplication (a*b) requires more mantissa bits than provided for by the floating point hardware.

20. (Currently amended) The computer-readable storage medium of claim 19, said floating point emulation algorithm further comprising:

determining whether the final resulting mantissa of the mantissa(a*b) - the mantissa (c) equals zero;

utilizing the floating point hardware to make the result of the calculation of a*b + c equal to zero when the resulting mantissa is equal to zero; and

calculating the remaining parts of the calculation of a*b + c using the integer hardware when the final resulting mantissa is not equal to zero.

21. (Currently amended) A system for providing a target computing environment comprising:

a target processor including floating point hardware and integer hardware; and

translator code for performing high precision emulation of program code instructions for a subject machine on a target machine, said translator code comprising code executable by said target processor for performing the following steps:

determining ~~if~~ that operands in instructions of the program code for the subject machine require a ~~different~~ higher precision than provided for by the hardware of the target machine; and

applying a floating point emulation algorithm to perform intermediate calculations on the operands of the instructions at a higher precision than the precision supported by the hardware of the target machine to produce intermediate values;

wherein at each stage, the intermediate values are tested to determine whether the intermediate values have reached a point at which the hardware of the target machine has enough precision to finish the calculation without loss of accuracy, such that when it is determined based upon the intermediate calculations that the floating point hardware of the target machine provides sufficient precision to finish the calculations required by the instructions without loss of accuracy, the floating point hardware on the target machine is utilized to finish the calculations; and the integer hardware on the target machine is utilized to perform calculations not selected to be performed by the floating point hardware.

22. (Canceled)

23. (Currently Amended) The ~~combination~~ system of claim ~~22~~21, wherein the program code instructions are accumulated instructions that are calculated at a higher precision than the operands capable of being handled by the target machine.

24. (Currently amended) The ~~combination~~ system of claim 23, wherein the program code instructions are floating point accumulated instructions of the form: $d = \pm(a*b \pm c)$,

wherein a, b, c and d are operands expressible as floating point numbers.

25. (Currently Amended) The ~~combination~~ system of claim 24, wherein said floating point emulation algorithm comprises identifying whether any of the operands (a, b, or c) are special values having a known result that all compatible hardware will produce regardless of the level of precision of said compatible hardware.

26. (Currently amended) The ~~combination~~ system of claim 25, wherein said special values include either zero, infinity, or NaN (not a number), wherein the floating point hardware is utilized

to calculate the result of the accumulated instructions when any of the operands (a, b, or c) are identified as special values.

27. (Currently amended) The ~~combination~~ system of claim 24, wherein said floating point emulation algorithm further comprises:

determining whether the exponent for the result of the multiplication of (a*b) overlaps with the exponent of c; and

utilizing the floating point hardware to calculate the result of the accumulated instructions when the exponent for the result of the multiplication of (a*b) fails to overlap with the exponent of c.

28. (Currently amended) The ~~combination~~ system of claim 27, wherein, when the exponent for the result of the multiplication of (a*b) overlaps with the exponent of c, said floating point emulation algorithm further comprises:

determining whether the mantissa for the result of the multiplication (a*b) requires more mantissa bits than provided for by said floating point hardware; and

utilizing the floating point hardware to calculate the result of the accumulated instructions when the result of the multiplication (a*b) does not require more mantissa bits than provided for by the floating point hardware.

29. (Currently amended) The ~~combination~~ system of claim 28, said floating point emulation algorithm further comprising computing the full calculation of a*b using the integer hardware when mantissa for the result of the multiplication (a*b) requires more mantissa bits than provided for by the floating point hardware.

30. (Currently amended) The ~~combination~~ system of claim 29, said floating point emulation algorithm further comprising:

determining whether the final resulting mantissa of the mantissa(a*b) - the mantissa (c) equals zero;

utilizing the floating point hardware to make the result <u>of the calculation of a*b + c</u> equal to zero when the resulting mantissa is equal to zero; and

calculating the remaining parts of the calculation of a*b + c using the integer hardware when the final resulting mantissa is not equal to zero.